Chapitre 1

Présentation du projet

Elaboration d'un moteur de rendu 3D simpliste en C++. Ce dernier, affichera à l'écran des objets 3D de type `Obj`, pour cela, il implémentera diverses méthodes de rendu :

- rendu `fil de fer`,
- rendu `flat shading`,
- rendu en utilisant la méthode `gouraud`,
- rendu avec la méthode `phong`,
- application de texture planaire sur les objets.

Bien entendu, nous gérerons l'éclairage pour donner un effet réaliste au rendu. Nous avons intégré une interface graphique utilisant la bibliothèque QT. L'utilisation du logiciel est ainsi plus intuitive.

Chapitre 2

Utilisation de visu

2.1 Les objets

Ouvrir un objet

Pour ouvrir un objet et donc l'afficher dans le mode de rendu courrant, il suffit de cliquer sur `Open` du menu `Object`.

Editer les propriétes de l'objet

Une boîte de dialogue permet de modifier les paramètres de l'objet. Nous pouvons donc régler la couleur (lorsque l'on n'est pas en mode texture) et les différents paramètres de réfléxion de la lumière (Ka, Kd, Ks et m).
On accède à cette boîte de dialogue par l'item `Material` du menu `Object`.

Appliquer une texture à l'objet courant

Pour ouvrir une texture et l'appliquer à l'objet, on clique sur `Texture` du menu `Object`. On passe automatiquement en mode texture planaire.

Déplacement de l'objet

Les déplacements de l'objet sont standard (ils se font à l'aide de la souris) :
- bouton gauche pour faire tourner l'objet sur lui-même (boule virtuelle),
- bouton droit pour translater l'objet.
- bouton du milieu pour zoomer.

2.2 L'éclairage

L'éclairage naturel : la lumière ambiante

Nous pouvons régler le coefficient de lumière ambiante grâce à la boîte de dialogue Light (item `Parameters` du menu `Light`). Ce coefficient doit être compris entre 0 et 1.

L'éclairage artificiel : les lampes

Grâce à la même boîte de dialogue que pour l'éclairage naturel, on peut régler l'intensité de la lampe et sa couleur.

2.3 Les modes de rendus

Le mode fil de fer (en vectoriel)

Ce mode de rendu permet d'afficher l'objet courant en mode `fil de fer` grâce à des primitives de tracé vectoriel. On accède à cette fontion par l'item `Wire` du menu `RenderModes  Vectorial`.

Le mode flat shading (en vectoriel)

Ce mode permet d'afficher l'objet courant en `flat shading`. Dans ce mode nous avons effectué un tri des facettes sur les faces visibles et grâce à l'algorithme du peintre on a éliminé les faces cachées. Un éclairage sommaire par rapport à la normale permet de donner des petits effets d'éclairage. Pour utiliser ce mode : item `FlatShading` du menu `RenderModes  Vectorial`.

Le mode fil de fer

Ce mode de rendu affiche l'objet en mode fil de fer. Contrairement au mode vectoriel, nous dessinons ici point par point directement dans une image. Item `Wire` du menu `RenderModes  Image`.

Le mode gouraud

Ce mode permet un lissage de l'objet grâce à l'algorithme de gouraud. Une lampe intéragit maintenant et permet de donner des effets réalistes. Item `Gouraud` du menu `RenderModes  Image`.

Le mode phong

De la même façon que pour le mode précédent nous appliquons à l'objet un algorithme de lissage : le phong. Cette méthode est beaucoup plus lourde que le gouraud et de ce fait ne permet pas de faire du temps réel. Bien évidement la lampe et l'éclairage ambiant interviennent. Item `Phong` du menu `RenderModes  Image`.

Le mode texture

Nous appliquons la texture courante à l'objet, le mode de rendu utlisé pour le lissage est le phong. Les coordonnées de texture sont calculées de façon planaire. Item `Phong` du menu `RenderModes  Texturing`.

2.4 La camera

Une seule fonction est implémentée pour le moment : le centrage de la camera. On la place de manière optimale par rapport à l'objet. On accède à cette fonction par le menu `Camera`.

Chapitre 3

Le projet visu

3.1 Les principaux types utilisés

Pour décrire les objets et leurs propiétés nous avons choisi la structure de donnée suivante :
Un objet est consitué de sommets et de triangles. Nous avons donc utilisé une structure `vertex_t` qui contient les coordonnées du sommet dans les trois repères (repère local, repère de la camera, repère du l'ecran en pixel), la couleur, la normale et enfin les coordonnées d'attachement pour la texture. Tous les sommets sont rangés dans un tableau, la structure `triangle_t` contient les indices des trois sommets formant le triangle en question.
En ce qui concerne le matériel de l'objet, la structure `material_t` permet de fixer tous les paramètres de réflexion de la lumière.

3.2 Déscription des classes

Pour tous les modes de rendu, nous ferons un test de visibilité par rapport à la normale pour déterminer les faces à afficher. Nous éffecteurons ensuite un tri de ces faces pour appliquer l'algorithme du peintre de façon à éliminer les faces cachées.

Classe `matrix.cpp`

Cette classe contient toutes les primitives permettant de manipuler et d'initialiser des matrices. Nous pouvons notamment créer des matrices à l'identité, des matrices de rotation pures et des matrices de translation. Nous avons surchargé l'opérateur `*` pour pouvoir faire des multiplications de matrices.

Classe `Application`

Cette classe permet de créer l'objet `QApplication` utilse pour l'interface graphique QT.

Classe `MainWindow`

Dans cette classe on intialise tous les objets drawer (les differents modes) et on traite les évènements souris et clavier.

Classe `Menu`

Cette classe implémente les menus de visu.

Classe `MaterialDialog` et `LightDialog`

Ces classes représentent les boîtes de dialogue permettant de paramétrer les propiétés de l'objet et les propiétés de la lumière.

Classe `Drawer`

Cette classe est virtuelle elle sera hérité de tous les drawer (un drawer par mode de rendu). Elle possède comme méthode virtuelle `DrawObject` qui permet d'afficher l'objet courant.

Classe `VectorialDrawer`

`ImageDrawer` hérite de la classe `Drawer`. Elle affiche l'objet en faces pleines grâce à la méthode `flat shading`. Elle implémente donc la méthode `DrawObject`. Nous utilisons ici des primites de QT permettant de remplir des polygones.

Classe `VectorialWireDrawer`

Permet d'afficher l'objet en mode fil de fer. Elle hérite de la classe `Drawer`. Pour dessiner l'objet nous utilisons des primites de QT permettant de dessiner des droites. On implémente ici la méthode `DrawObject`.

Classe `ImageDrawer`

`ImageDrawer` hérite de la classe `Drawer`. Toutes les classes qui utiliserons une image hériterons de cette classe virtuelle. Nous dessinerons point par point directement dans cette image en 32 bits. Cet image est en fait un tableau qui contiendra toutes couleur (r, g, b) que l'on verra à l'écran, ce tablau a donc pour taille la résolution de la caméra (resx * resy).

Classe `ImageWireDrawer`

Permet d'afficher l'objet en `fil de fer`. Cette classe implémente `DrawObject` et hérite de `ImageDrawer`.

Classe `GouraudDrawer`

Implémente la méthode `DrawObject` et hérite de `ImageDrawer`. Cette classe permet donc de construire des drawer qui utiliseront le lissage de gouraud pour afficher les objets.

Classe `PhongDrawer`

Implémente la méthode `DrawObject` et hérite de `ImageDrawer`. Nous utiliserons ici la méthode phong pour dessiner l'objet en cours. Dans cette classe, on calcule directement l'influence de la lumière sur l'objet. Le précalcul est ici impossible du fait que phong clacule une interpolation des normales pour effectuer le lissage.

Classe `TextureDrawer`

Implémente la méthode `DrawObject` et hérite de `ImageDrawer`. Nous calculons dans cette classe les coordonnées d'attachement 2D u et v. Ces coordonnées sont calculées de façon planaire. Pour le lissage, on utilise la méthode phong.

Classe `Entity`

Cette classe représente des entités. C'est à dire un objet évoluant dans le monde (ex : cube, camera, light, ...). Toutes les méthodes de deplacement sont implémentées ici (translation, rotation).

Classe `Camera`

Hérite de la classe `Entity`. La caméra est une entité particulière, elle possède des paramètres propres à une camera. Elle possède une méthode qui permet de calculer la valeur des points de l'objet à l'écran.

Classe `Object`

Hérite de la classe `Entity`. Représente les objets que le renderer devra afficher. Cette classe possède des méthodes permettant de gérer l'objet en fonction de la camera, notamment placer la camera de manière optimale.

Classe `Light`

Hérite de la classe `Entity`. Cette classe implémente toutes les méthodes permettant de calculer l'effet de la lumière sur les objets.

Classe `BBox`

Représente la bouding box associé à un objet.

Classe `Texture`

Permet de créer un objet `Texture` à partir d'un fichier `rgb`.